3. Model and Framework for Application Network Deployment on a Programmable Internet Service Infrastructure

As it has been shown in chapter 1 there is a need not to create application networks activating application servers uncoordinatedly. Application networks must be deployed "to spread out or arrange for effective action". Reviewing state of art, we found how application networks are coordinated, and which programmable infrastructures exist where an application network might be deployed. However there is no system that deploys in a coordinated manner an application network.

This chapter describes first, a model for application network deployment and second, a framework for application network deployment in a programmable Internet service infrastructure.
3.1. Deployment Model

Many computer science problems are solved using metaphors from real world systems that are abstracted into a model. This conceptual model is later translated into an implementation. Deployment in a process that takes place in the physical world; many real world organizations and structures are deployed. Deployment takes place in the physical world when new human organizations have to be created: armies have to be deployed, sales forces have to be deployed, game teams have to be deployed, etc. Deployment is motivated by the existence of buyers, enemies, rivals; which stimulates, encourages, demands rapid construction, formation of sales forces, armies, teams. Since resources required, sales points, safe positions, strategic positions are expensive and scarce, it is necessary to coordinate them in a planned manner for effective action. This problem has been solved in different engineering and science fields: military logistics, sales force planning, game strategy, etc., by implementing of a deployment plan. Those deployment plans are composed of several tasks: resource allocation, salesmen, soldiers, players transportation and divisions, troops or teams coordination for effective action.

Fig 3.1. Sales Forces, Army, Teams Deployment
As well deployment takes place in the physical world when complex physical structures are have to be created: hospital deployment, oil stations deployment, euro deployment, etc. Deployment is motivated by the existence of ill populations, automobile users, euro-consumers, etc. which stimulates, encourages, demands rapid construction, formation of hospitals, oil stations, euros. Since resources required building space, are expensive and scarce, it is necessary to coordinate them in a planned manner for effective action. This problem has been solved in different engineering and science fields: urban planning, hospital location, and industrial logistics, by implementing of a deployment plan. Those deployment solutions involve: allocation resources, distributing hospital equipment, building material, euros and coordinating hospitals, oil stations, exchange points, etc for effective action.

Fig. 3.2. Hospitals, Oil Stations, Euros Deployment
3.1.1. Model for Deployment of Application Networks

Abstracting those deployment solutions, we propose a deployment model for computer application network. In this deployment model there are application users which demand a service (fig. 2.a); a limited number of resources with which to construct that service (2.b); a deployment plan determining which resources have to be allocated, how to coordinate service instances and how to build the service (2.c); this deployment plan is executed by allocating resources (2.d), distributing service code (2.e) and installing and coordinating servers which provide the application (2.f).

A deployment plan is designed to accomplish two different objectives: 1) provide requested service level, and 2) minimise resource utilization, while fulfilling any other imposed constraints. A deployment plan specifies application network deployment.
commands. The service plan contains all commands and information to activate or modify an application network, including resource allocation commands (which resources have to be allocated), and service composition commands (how to distribute code, how to bind services to resources, and how to coordinate application network servers).

For each service demand/resource offer pair there can be several deployment plans, corresponding to different deployment strategies; i.e. it can be chosen mapping algorithm values, preferred service areas, modification interval, type of resources to allocate, allocation mechanism, code distribution mechanisms, etc.
3.2. Framework Building Blocks

To realize such model we propose to implement a framework. A framework is an extensible implementation of a model. A framework is composed of several building blocks that can be extended independently of each other. Such framework will permit future applications to be deployed reusing existing building blocks. Building blocks are designed so as to be as generic as possible so that minor modification will be required for future new application deployment.

Building blocks of the framework are:

- **A programmable Internet service infrastructure**:  
  Comprise a number of nodes that provide resources and an execution environment. These nodes must permit simultaneously operation of several servers belonging to different application networks.

- **Resource discovery and monitoring mechanisms**:  
  Goals of these mechanisms are to obtain resource availability. Resources are unknown at initial time, having to be discovered. Besides later new resources can become available, also having to be discovered. Resources monitoring must take place to verify its availability and changes in characteristics and quantity.

- **Service specification interface**:  
  So that service providers input their requirements from their expectations of service demand. Or to modify them afterwards based on actual service performance. Service specifications summarize service demand and include service-wide properties and resource requirements, which indicate how the overall service is provided to users.

- **Resource mapping**:  
  To match service specifications with resource offer, service demand and service-wide constraints must be translated or mapped to a resource offer. Resource mapping is the translation from high-level service specifications to low-level resource requirements,
which includes the selection of those resources that best match the service requirements among all those available.

- **Deployment plan creation and control:**
  If enough nodes have been found, a deployment plan is created containing every command to be carried out for application network deployment. Such plan has to be completed in a controlled manner, detecting any error operation and reverting to a previous state if necessary.

- **Resource allocation:**
  Involves putting apart resources required at different nodes for sole used by a service.

- **Code distribution and service composition:**
  Once resources are allocated, services can be composed on those resources with service code distributed to those nodes. At each node application servers has to be bound to local resources, and configured to communicate and coordinate with other instances.

  Resource discovery and service specifications provide input to create a deployment plan with a resource-mapping algorithm. Allocating resource, distributing code and composing an application network in the programmable infrastructure execute the deployment plan. Changes in demand or resource availability are feedback to resource discovery and service specification modules triggering a re-deployment operation. Besides threats can attack resources, communications and applications. Measures have to be implemented to eliminate or paralyse those threats.
Fig. 3.4. Framework building blocks
3.3. Framework Architecture

Architecture is derived from roles appearing in the model at section 3.1. The deployment model clearly has two differentiated roles: service provider and resource providers. Service providers assume an active role in that they command deployment of the application network. A deployment manager, which creates deployment plans and commands resource allocation and service composition, represents them. Resource providers are programmable infrastructure nodes. They assume a passive role in that they provide resources and an execution environment to set up a service. A resource agent, which will follow deployment managers’ commands, represents resource providers. The application network deployment framework implementation requires system architecture composed of resource agents at resource providers’ nodes and deployment managers at service providers' nodes. Deployment managers interact with resource agents to deploy application networks.

Fig. 3.5. Deployment managers and resource agents attached to service providers (SP) and resource providers (RP).
Deployment managers and resource agents implement framework building blocks: resource discovery and monitoring, service specification, resource mapping and allocation, and code distribution and service composition. Deployment managers discover resource agents, and query them about its resource capabilities. Deployment managers receive deployment request from application providers, create a deployment plan by performing resource mapping. Deployment managers command resource agents to allocate resource, obtain service code, bind service programs to allocated resources, and configure application network coordination. Resource agents on deployment managers commands perform those commands and resource agents respond whether it was successful or not.

Fig. 3.6. Interactions among: service provider --deployment manager--application --resource agent -- resource—resource provider
3.4. Building Blocks Description

3.4.1. Programmable Internet Service Infrastructure

As seen in chapter 2 there exist several programmable infrastructures. Those programming infrastructure provide a large pool of resources distributed throughout the Internet, where services can be executed. They differentiate from distributed operating system because they take into account the heterogeneity, scale and security challenges of Internet environment, by implementing mechanisms that allow for third party service activation, and by providing mechanisms for code distribution. However they only provide interfaces for activating services in a single node, either generic Unix applications as WebOS [Vah98] and ejasent [EJA00] or very specific services: active network services alan [Fry99] and anetd [Ric98a], or active services [Ami98][Cao98]. Even some proposed infrastructures provide only storage resources for storage demanding services: Distributed Storage Infrastructure DSI [Bec98], Intelligent Storage Market [Chu99] or the logistic backbone Lbone[Bas02].

The main characteristic of a programmable Internet service infrastructure is that it must allow multiple application networks to coexist. It must provide resources distributed throughout the Internet. And it must permit multiple virtual application networks to be executed simultaneously, sharing those resources. Due to programmable, new Internet services can be dynamically activated and coordinated with other instances. Therefore an Internet service programmable infrastructure has three characteristics that permit several application networks to coexist on it: shared resources, virtualisation and programmability, as represented in figure 3.7. These are explained next.
Fig. 3.7. A programmable Internet service infrastructure with multiple application networks sharing resources in different execution environments.

Shared Resources

Infrastructure provides resources that Internet servers programs require for its execution. Resources required by application network servers are storage, network connectivity, and processing power. Each application server requires storage to store its code and application data; it requires processing power to serve clients request; and it requires network capacity to transmit client responses and for inter-server communication.

Resources are defined by its properties and its constraints. Network resources main property is its adjacent regions where service can be provided and latency to those regions. Storage resources main properties are its read and write access time. Processing power main property is processor architecture and speed.

Each individual resource node has its own constraints that conditions which services can be mapped to it. Maximum network capacity at service nodes limits maximum service traffic that can be provided from that node by any single service and by all services in that
node. Maximum storage size in a server prevents services with higher storage requirements from being serviced there.

<table>
<thead>
<tr>
<th><strong>Resource Type</strong></th>
<th><strong>Resource Properties</strong></th>
<th><strong>Resource Constraints</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Inet conectivity</td>
<td>Id (IP)</td>
<td>Max BW (Mbits)</td>
</tr>
<tr>
<td></td>
<td>Nearby Regions (#AS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Latency to region (msec)</td>
<td></td>
</tr>
<tr>
<td>Storage</td>
<td>Access time (msec.)</td>
<td>Max Storage (Mbytes)</td>
</tr>
<tr>
<td>CPU</td>
<td>Type (586)</td>
<td>Max Load (%CPU usage)</td>
</tr>
<tr>
<td></td>
<td>Speed (Flops)</td>
<td></td>
</tr>
</tbody>
</table>

*Table 3.1. Resource Properties and Constraints*

**Virtualization**

Infrastructure has to permit several application servers to be executed simultaneously. But each server has to think it is alone and it can use all available resources. This is implemented by a virtualisation mechanism, which permits nodes to share its resources, without applications being aware of. As seen in figure 3.7., in node mosaic.ac.upc.es there are three applications executing, two executing at TOMCAT execution environment, and one executing as a Linux server. Each application is not coded to take into account that there are other applications being executed at the same node. The execution environment takes charge of it, so that they do not interfere with each other. Virtualisation support for multiple application networks coexisting on the same infrastructure requires each node to have at least: a) a multitasking execution environment that share CPU usage among different applications, b) a virtual memory manager which shares real memory among different applications and c) a network abstraction that supports simultaneous multiple network connections. Multitasking and virtual memory are two abstraction widely researched and supported by many systems. Support for multiple simultaneous network connections can be implemented at the network layer by multihomed hosts, hosts with multiple IP addresses, which requires a minimal
routing functionality [Wa02]. Or TCP and UDP port abstractions can be used to create several application layer connections.

Virtualisation mechanisms have a higher and a lower limit of simultaneous services being concurrently executed. The higher limit is imposed by the fact that each service has some set up cost associated. Therefore if a node provides too many services, there will be bad resource utilization. The lower limit is imposed by the fact that if a low number of applications is being provided by a node there can be a large percentage of resources unused in such node. This limit can be determined by a maximum resource usage per service at each node. If it is not very high, a node will provide a large number of services. Beside it has the advantage of compelling applications to request resources in several nodes, increasing its load balancing and fault tolerance.

<table>
<thead>
<tr>
<th>Virtualization Constraints</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. number Service</td>
<td>Else high set up costs</td>
</tr>
<tr>
<td>Max. resources per service (Min. num. Service)</td>
<td>Else unused resources</td>
</tr>
</tbody>
</table>

*Table 3.2. Virtualization constraint*

**Programmability**

Infrastructure has to provide an execution environment where application programs execute. An execution environment provides a set of basic primitives for access and manipulation of local resources. An application program is coded with calls to a set of basic primitives; therefore it is coded for execution in a particular execution environment. Execution environments can be tied to a particular hardware, thus application programs are also coded for a particular hardware. Some execution environments provide the same set of primitives for different hardware platforms; applications are written for a particular execution environment and later compiled for execution in a particular hardware. If application network server programs are provided in binary format, they can only be executed in nodes that have the
same processor and operating system. If application programs are provided in source code, execution environments have to provide tools for compilation.

Besides each node can provide several execution environments. Execution environments are different means of accessing local resources. Provided that mechanisms for resource scheduling among different execution environments exist, application executing in different execution environments can also coexist concurrently.

<table>
<thead>
<tr>
<th>Execution Environment</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of EE</td>
<td>Type (Linux-i586, TOMCAT)</td>
</tr>
<tr>
<td></td>
<td>Compilation Tools</td>
</tr>
<tr>
<td></td>
<td>Libraries</td>
</tr>
</tbody>
</table>

*Table 3.3. EE Properties*
3.4.2. Resource Discovery and Monitoring

To assign resource for application network deployment, it is required in first place to identify existing resources and its properties. In second place, it is required to keep track of its availability. Resource offer changes over time due to resource additions and removals from the system at resource providers will or due to failures, and due to resource allocation and reallocation.

Resource discovery and monitoring provides deployment nodes with an up-to-date resource offer specifying which nodes support activation of new services, its resource type and quantity. But resources have not to be discovered or monitored directly by deployment nodes. Deployment nodes are only interested in discovering resources which they can exercise control, therefore it makes sense to discover only agents which control some resources. Therefore resource discovery is simplified to agent discovery. Search algorithms applied to computer networks can realize resource agent discovery. Overall resource discovery and monitoring functionality has to follow the scheme of figure 3.8. Methods used by resource agents to monitor resources are represented: polling, notification and probes and inference, and will be explained in next section.

![Diagram of Deployment Nodes, Resource Nodes and Resources](image)
Resource Availability by Polling, Inference and Notification

Resource agents are bound to some resources and execution environments whose status and properties they have to be informed. However each type of resource requires specific mechanisms for monitoring its status and properties. Existing resource monitoring techniques make use of polling mechanisms, notification mechanisms or inference mechanisms. Polling mechanisms are based on agents periodically requesting status to resources; notification mechanisms are based on resources notifying agents of status changes. Inference mechanisms are widely used in the Internet because of its layered architecture: low layer resources do not make its status information available to higher layer entities, therefore entities have to infer lower layer resources and status by sending probes and analysing its effect. Network probes are used to infer network topology, network delays or others. Computer contained resources such as storage or processing power are monitored resource by system specific mechanisms, implemented by local operating systems.

Discovery and Monitoring of Resources Agents

Every deployment manager has to obtain such resource availability information. Initially resource availability information requires a resource agent discovery phase. Resources agents can be discovered and monitored proactively (managers query nodes) or indirectly (managers lookup resource agents, and resource availability, in a directory updated by resource agents, f.e. Globus MDS [Cza01]). There are many alternatives to discover nodes proactively. We consider two methods that have been reported to discover a large number of nodes. These are: address space scanning [Gov00] and multicast expanded ring search [Moy94].

To maintain such information up-to-date there are two extreme solutions, initiating a new resource discovery when a new deployment request arrives, implemented by Xbone, or permanently monitoring every resource node. The former method does not require a resource
availability table. However resource information is not immediately available when required, there is a resource discovery time proportional to network diameter. The latter method has the advantage of faster local access to such information, however at the expenses of maintaining a table of resource availability information and creating monitoring traffic. Xweb implements this second mechanism.

**Multicast expanded ring search**

Multicast expanded ring search is a cost-effective option for discovery of network nodes in multicast enabled networks. It sends increasing time-to-live discovery messages to a multicast address, reaching further apart nodes. Nodes receiving those messages can notify its presence and properties to sending node. This mechanism, sketched in figure 3.9., should discover large numbers of nodes in a short time with little network overhead.

![Discovering increasingly further apart nodes with Multicast Expanded-Ring Search](image)

**Address space scanning**

This technique can be used in non-multicast enabled networks. It has been implemented in tools such as Mercator[Gow00]. Besides it provides good results for discovery of Internet topology. The main mechanism of such techniques is based on address space scanning.
Different addresses of the searched address space are probed by sending initial messages of searched agents protocol, existing agents will respond as if initiating a communication. Not to make a random scanning which can be very costly, some heuristics must be employed. The basic heuristic employed by Mercator is to probe first addresses of local networks which have the same IP prefix as the discovery manager, them traceroute is used to discover addresses of adjacent networks, its prefix is extracted and all addresses with that prefix are probed. Address space to be scanned is composed of all possible address of nodes. These addresses are composed of an IP address plus TCP port number assigned to service to be discovered. In figure 3.10, it can be seen how node D discovers all nodes in its subnetworks and its adjacent subnetwork.

*Fig. 3.10. Discovering resources agents with address space scanning*
3.4.3. Service Specification

Specifications of an application network service are used to determine which resources have to be allocated and how to construct and coordinate a service with those resources. In its general case a service specification is made up of three different elements: node requirements, service demand and service constraints. Node requirements are characteristic of each service, independent on the number of clients: examples are code and data size, execution environment type, per client network traffic, per client inter-servers traffic, and per client service level expressed by maximum client-server distance, or maximum server load. Service demand is independent of node requirements, it is the expected clients demand for a service; it represents clients population, clients locations and access frequency of clients. Service constraints impose some condition on the whole service, such as maximum diameter, level of redundancy, maximum cost, minimum service level or maximum resource utilization.

Specifications of network flows [Aur98b] and server requirements have being proposed by several groups. Chuang [Chu99b] has proposed specifications for distributed storage requirements of an application; however their characterization simply modifies network flows QoS specifications and classifies applications by their caching constraints. Gongs et al [Su00] describe specifications of application networks as modifications of virtual private networks specifications, thereby not taking into account some important properties of application networks. There is one main difference between layer 3 networks and layer 7 network, which conditions how to define its specifications: whereas layer 3 data networks demand is determine by a table stating how much traffic flows between any sources and destination, in an application network a single service source is expected, therefore a traffic table can be represented by a single array representing the demand of each region.
Application Networks Specifications

Application networks have different characteristics from other type of services, which are summarized in table 3.4. Node requirements should be normalized for one client demand. Examples are storage size, execution environment, which defines which execution facilities are required by a service for its execution, it can be expressed by an Operating System type, though it can be include specific requirements for a processor type, support for floating point operations, etc. Per request network traffic represents average network traffic created by each request. Inter-servers traffic, again as per client, represents average traffic generated between two servers for each client’s request. Per client request duration is average duration of a client’s request. Server load represents maximum expected server load, highly loaded server can enter into congestion regions easily, where service level cannot be guaranteed. Service level is determined by client-server distance. The higher the client to server distance the worse service level can be expected. Client to server distance can be measured by response time, Inet hops or path throughput, congestion level.

Service demand is independent of service characteristics; represent clients demand for a service. The service provider, from his expectations of future service demand, can provide service specifications at initial time. Or can be inferred from past service demand logs. It represents clients’ population and access frequency of each client. There is a demand traffic profile to be met; each client regions will request an amount of service traffic depending on the demand of the service in that area. A service traffic table that indicates how much traffic capacity each service region requests determines demand. This demand matrix states how much service traffic is provided to an Internet region. There are several possible tables for each service depending on how those regions are interpreted; regions can be IP subnets, AS areas, or geographical areas.

Service-wide constraints are: maximum diameter, which represents how many servers a request will be forwarded to before being provided by the origin server. High diameter
Application networks provide the least costly services because few requests reach to distance
origin servers, however low diameter application networks can provide lower response times
since every forward operation adds a delay. The level of redundancy aims at increase the
service availability of service; it indicates how many different servers would service any
client or any edge server. Though a service node could provide for all service traffic in a
given region in the most cost-efficient way, it is not the best solution in terms of reliability, if
this node fails there will not be a back up server. In a similar way edge servers should be able
to forward a request to different servers. Maximum cost, though service deployment tries to
achieve cost-effective allocations some times the least costly allocation can exceed the cost
the service deployers are ready to pay, therefore a maximum cost can be specified for a
service before being deployed.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Comment</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>EE:</td>
<td>Per node</td>
<td>Linux, NT, JVM,...</td>
</tr>
<tr>
<td>Processor type:</td>
<td>Per node</td>
<td>I386, Alpha, ...</td>
</tr>
<tr>
<td>Storage:</td>
<td>Per node</td>
<td>200 Mbytes,...</td>
</tr>
<tr>
<td>Processing power:</td>
<td>Per client session</td>
<td>8000 Instructions,...</td>
</tr>
<tr>
<td>network traffic:</td>
<td>Per client session</td>
<td>1Mbit, 100 Kbit, ...</td>
</tr>
<tr>
<td>inter servers traffic:</td>
<td>Per client session</td>
<td>10Kbit, 1 Kbit, ...</td>
</tr>
<tr>
<td>Session duration:</td>
<td>Per client session</td>
<td>10 sec., 1 hr., ...</td>
</tr>
<tr>
<td>Service level:</td>
<td>For all sessions: max RTT, min BW, ...</td>
<td>10 msec</td>
</tr>
</tbody>
</table>

**Demand Matrix**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Comment</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regions:</td>
<td>List of #AS, IPs, ...</td>
<td>AS #1210, IP:137.0.0.0, ...</td>
</tr>
<tr>
<td># Clients:</td>
<td>At each region</td>
<td>1-10000</td>
</tr>
<tr>
<td># Sessions:</td>
<td>Avrg. per Client</td>
<td>1-100 req/sec</td>
</tr>
</tbody>
</table>

**Service Constraints**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Comment</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectivity:</td>
<td>Topology graph.</td>
<td>1-level hyrarchy, spanning tree,...</td>
</tr>
<tr>
<td>Coordination Rules:</td>
<td>Per node or per service</td>
<td>Forward to, redirect to server Y,...</td>
</tr>
<tr>
<td>Maximum diameter:</td>
<td></td>
<td>3,6,...</td>
</tr>
<tr>
<td>Level of redundancy:</td>
<td>Min # servers, Max # clients per server</td>
<td>1, 2, 3,...</td>
</tr>
<tr>
<td>Maximum cost:</td>
<td>Max # servers</td>
<td>8 servers</td>
</tr>
</tbody>
</table>

*Table 3.4. Application network specifications*
Specifications Notation

Service specifications have to be constructed in a way so that every party involved in the process of deploying a service knows exactly what to do to create the service as it is being requested. Service specifications for application networks can be expressed on different notations: from a full programming language which could incorporate service code as part of the specification, to a mark-up language that provides functionality to express data according to some predefined data structures which can be processed automatically by applications. The most used mark-up language is XML [Con97]. An application network specification in a mark-up language, XML, following an application network template is shown in figure 3.11.

```xml
<app_net_specifications id=www.mortero.com
 Origin_Server="127.10.12.12:80/mortero.tar">
  <Node Requirements>
    <OSType>SunOS</OSType>
    <Storage Unit="Mbytes">200</Storage>
    <Service_Traffic Unit="Kbits">20</Service_Traffic>
    <Service_Time Unit="sec">10</Service_Time>
    <Client_Distance Unit="InetHops">2</Client_Distance>
  </Node Requirements>
  <Service Demand>
    <RegionalDemand>
      <RegionId Unit="ASnum">1201</RegionId>
      <ClientsNum>1000</ClientsNum>
      <DemandRate Unit="req/sec">0.1</DemandRate>
    </RegionalDemand>
  </Service Demand>
  <Service-wide Constraints>
    <Max_Network_Diameter>1</Max_Network_Diameter>
    <Redundancy_Level>2</Redundancy_Level>
    <Max_num_servers>5</Max_num_servers>
  </Service-wide Constraints>
</app_net_specifications>
```

Fig. 3.11. Application Network Specifications
3.4.4. Resource Mapping

Resource mapping is defined as the translation from high-level service specifications to low-level resource requirements. The mapping problem is finding a function that translates from requirements expressed in a high layer metrics to resources defined in a low layer metric. In QoS network architectures [Aur98b] we can find many examples of which are the difficulties of mapping those high level requirements to CPU, memory and I/O resources.

The resource mapping function for application networks must choose among those resource sets that conform to service specifications, in term of service level provided and service-wide properties, that one with lower resource utilization. Resource utilization in an application network comprises storage resources, proportional to the number of service points multiplied by resource used at each node; processing resources, proportional to clients demand; and network resources, composed of client to server traffic plus inter-server traffic, as well proportional to clients demand.

Resource mapping functionality chooses the most appropriate nodes where to deploy application networks. This selection is made based on these restrictions: 1) resource characteristics and availability, 2) service specifications and 3) optimising some value. Resource availability determines which set of nodes are candidates. Some nodes are not chosen as candidates because they do not provide resources required by applications, or its resources have some constraint, which is not compatible with this application. Service specifications reduce the number of candidates and some combinations can be discarded, either because they do not provide enough resources for the demand, or because service wide constraints are not met by such a combination. For example in figure 3.12. two out three possible mapping are discarded because they do not fulfil resource requirements (not enough resources at some node), demand regions (no resource at some region), service constraints (maximum diameter, maximum number of servers). Finally among remaining mappings, it is
selected the one which optimises some value: either maximising overall service quality (service quality is measured by client-to-server distance and/or server load; distance is proportional to response time, Inet hops or path throughput; the higher client-to-server distance the worse service level can be expected) or/and minimising costs (cost is measured by resource consumption per server cost plus network traffic).

Fig. 3.12. Different mappings of application network specifications

<table>
<thead>
<tr>
<th>Application</th>
<th>Execution Environment</th>
<th>Network Capacity</th>
<th>Storage Capacity</th>
<th>Service Request</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Transfer</td>
<td>Linux-302, 70GSDT</td>
<td>10</td>
<td>2000</td>
<td>1510, 1710, 1910</td>
<td>On-Demand</td>
</tr>
<tr>
<td>File Transfer</td>
<td>Linux-303, 70GSDT</td>
<td>10</td>
<td>2000</td>
<td>1510, 1710, 1910</td>
<td>On-Demand</td>
</tr>
<tr>
<td>File Transfer</td>
<td>Linux-304, 70GSDT</td>
<td>10</td>
<td>2000</td>
<td>1510, 1710, 1910</td>
<td>On-Demand</td>
</tr>
</tbody>
</table>

Per node Resource Mapping

Per client application specifications are represented by: transaction download time, request rate, number of instructions per request, and minimum service level. It should be translated to host resources defined by processing rate, storage capacity and networking resources [!!]. However it is not a straightforward task. Providing a service demand of 100 request / sec for file transfer of 1Mbytes files in less that 800 sec. will require different computational, storage and network resources depending on code implementation, and host architecture, and machine load. Though we can observe that at least it will be required network resource of 1
Mbits capacity, and storage resource of 1 Mbyte storage capacity. A performance model of each resource node for each application should be obtained to effectuate a correct mapping.

<table>
<thead>
<tr>
<th>Service</th>
<th>1 Mbytes file download</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; 800 sec. service time</td>
</tr>
<tr>
<td></td>
<td>100 req/sec</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resources</th>
<th>1 Mbit network connection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Mbyte Storage</td>
</tr>
</tbody>
</table>

Fig. 3.13. High-level service specifications mapping to low-level resources

“Connected Placement” Mapping Algorithm

In the application networks literature there can be found several algorithms that optimise application networks, subject to different constraints. They differentiate on which system parameter they optimise, and constraint resources. Some considers storage as the unique constrained resource and try to minimize overall client-to-server distance [Chu99b][Rad01], or maximize hit rates [Kor99], other consider server bandwidth is constraint [Ven01]. Others model application networks as a load-balancing problem where a certain server load threshold should not be surpassed, and minimize replication cost and client-to-server distances [Rab99]. A bit apart, virtual layer 3 networks consider network bandwidth as the most constraint resource and implement tree-minimizing algorithm where end-to-end distance is optimised [Isa01].

But application networks are constraint by three types of resources: storage, proportional to number of service points multiplied by resource used at each node, processing,
proportional to clients demand, and network, composed of client-to-server traffic plus inter-server traffic, as well proportional to clients demand. Besides they must optimise C/S distance and total traffic created.

<table>
<thead>
<tr>
<th>Mapping Algorithms</th>
<th>Constraint Resources</th>
<th>Optimized Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placement</td>
<td>Storage (Mbytes)</td>
<td>Min C-S distance</td>
<td>Stor-Serv</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Server Load (Req/sec)</td>
<td>Min Network Traffic, Min C-S distance</td>
<td>Radar [Rab99]</td>
</tr>
<tr>
<td>Min Tree</td>
<td>Network BW(Mbits/s),</td>
<td>Network Traffic, e2e diameter</td>
<td>Vandal VPN provision[Isa01]</td>
</tr>
<tr>
<td>“Ideal”</td>
<td>Load, Storage, Network (Req/sec, Mbytes, Mbits)</td>
<td>C-S distance, e2e diameter</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5. Mapping algorithms

We have designed a simpler algorithm that we call “connected placement” application network resource mapping algorithm. We consider application networks create most of its traffic from edge servers to clients (caching can reduce outbound traffic by 40%, therefore it is a reasonable assumption). Application networks can be considered as "edge-resource demanding" vs. "core-resource demanding” services such as layer three IP networks. A core-resource demanding service aggregate resource requirements at the infrastructure core, whereas an edge-resource demanding service requires more resources nearer to clients because it caches responds at the edges. For example in figure 3.14, a layer three overlay might demand 10 Mbit network bandwidth for each client, therefore core bandwidth requirement shall be 60 Mbits, whereas an app network with equal end user requirements will require only 1 Mbits core bandwidth, if proxy caches have 90% byte hit rate.
This high difference makes inter-server traffic irrelevant for minimal cost search, and simplifies the problem to finding those edge nodes that minimize client to server traffic. This calculation is an optimisation problem solved by graph theory called placement algorithms, which were first proposed by Hakimi [Hak64]. The mathematical resolution of this problem has already been well studied to optimise other type of systems. This problem or its variants have appeared on different engineering fields: urban planning, hospital location, industrial logistics, commercial planning, military logistics, etc. Several solutions have been proposed for problems with different specification constraints.

```plaintext
resource_mapping {
    demand_regions = client_regions;
    do {
        server_locations = locations_minimize_distance (locations, demand_regions);
        demand_regions = regionsof(server_locations);
        current_diameter++;
    } While (current_diameter < max_diameter)
}
```

**Fig. 3.15. Connected placement algorithm**
Since client-to-server traffic shall be proportional to client-to-server distances, positions of edge servers can be calculated by a placement algorithm which minimizes overall client-to-server distances selecting only among those nodes with appropriate resources available. In a second phase those edge servers locations can be consider as client locations of second level servers, therefore its locations can be found executing the placement algorithm again, and this steps is repeated until the maximum network diameter is reached, when servers are fully connected to each other setting up a mesh.

To determine non-edge servers in an application network requiring an n level hierarchy, the placement algorithm must be executed iteratively n times, where each recursion uses previously mapped locations as initial demand set. As shown in figure 3.16. a service with a specified demand of clients from regions c1, c2 and c3, plus a two level hierarchy as request forwarding organization; in a) this service demand can be mapped to two resource sets; in b) executing the placement algorithm, resource set composed of r1, r2, and r3 is selected; in c) taking selected edge servers as demand locations for core server placement, two different resource mappings are possible; in d) executing the placement algorithm one of these mappings is chosen.
**3.4.4. Deployment Plan Creation and Transactional Control**

If resource mapping finds enough and appropriate resources, it is created a deployment plan containing every command to create the application network. Group creation mechanisms have been implemented previously by Piranha [Muf97] to explicitly create a group in Horus group communication systems [Ren96]. The main requirement of such operation is that execution of deployment plan has to be an atomic operation. Commands are executed in several remote nodes that can fail independently: they might not be able to allocate resources, or they cannot activate an application, etc. A mechanism has to be implemented so that a deployment operation takes place in all or none resource agent. The solution to such a problem requires a transactional monitor who controls every operation sub-process, in the case any single sub-process failure, every other sub-process is cancelled [Gra93].

With this goal deployment managers kept a deployment process control table for performing such operation transactionally. For each application which has been requested its deployment, there is a table where status of commands sent to resource agents is kept: command sent, command being processed, command cancelled because not enough resources, command cancelled because could not activate application, etc. If some operation is not carried out some alternative actions have to be performed: to rollback deployment,
canceling all actions carried out so far, or to perform resource mapping again to find substitutes to failing nodes.
3.4.5. Resource Allocation

Resources are scarce; therefore they must be shared for coexistence of several applications at each node. However if too many applications share a resource, slices are so small that become useless. Some mechanism is required so that applications have a minimum set of allocated resources to provide a minimum service quality. Resource allocation presents two problems: how to allocate resources remotely at several nodes, and how to allocate resources locally at each node among different applications.

Remote Multi-node Resource Allocation as Atomic Operation

Remote resource allocation requires a protocol for remote allocation between a resource allocator and several remote resource nodes. There are a few protocols for remote resource allocation: RSVP (Resource reSerVation Protocol) [Zha93], which has been proposed for bandwidth allocation at routers for network sessions. Or GARA [Fos99] used for CPU allocation at computational grids. Xbone allocates link interfaces to set up network overlays. These protocols have different characteristics because they have different goals. RSVP is designed to allocate resource along a number of routers for real time conferencing. GARA is design to allocate in advanced or on demand computational resources in a number of nodes distributed thorough the Internet.

Application network deployment requires resource allocations in a number of nodes distributed throughout the Internet, similarly to GARA, a centralized allocator can command local resources manager. However it has to allocate different types of resources (bandwidth, CPU and storage) for shorter time intervals, in addition those nodes need to be connected and coordinated. Xbone overlay deployment protocol, which allocates resources in nodes
distributed throughout the Internet that have to be connected and coordinated, is an appropriate protocol that can be extended for allocation of different resources.

**Local Resource Allocator**

A local resource allocator resolves conflicts among allocation request from different remote allocators. A local resource manager at each resource node will keeps a list of local allocated resources so that it does not allocate more resources than available. Local resource managers control that maximum quantity of resources is not exceeded. Also they control that allocations do not demand an allocation smaller that minimum resource allocation quantity. There is a cost associated with each allocation operation, if allocations are very small such cost can exceed benefit of allocations. Besides allocation tables can become very large compromising scalability as it happens with RSVP [Gue97].

Once resources are allocated and bound to applications, it has to be controlled that applications only use resources allocated to them. It should enforce strict partitioning of resources so that applications being provided in one node do not interfere with each other. Either using resources allocated to other applications, or consuming more resource that they were allocated.
Xweb Resource Allocation Deployment Manager Resource Agent Storage Mgr

ServiceName= cacheservice.companyA
ServiceStorage= 200 Mbytes
ServiceBW= 1 Mbit

Allocate 200Mbytes 1 Mbit OK OK

Fig. 3.17. Xweb Remote Multi Resource Allocation Protocol and Local Resource Allocator
3.4.6. Service Composition

Application network services are composed of software entities executed at different nodes. These programs make use of allocated resources to provide services and they are coordinated for effective service and efficient utilization of resources.

Dynamically composing a service involves binding those software programs to allocated resources and coordinating different service instances. The mechanisms required to compose an application network, shown in figure 3.18, are: 1) to distribute service code to nodes where it should be activate, 2) to bind service code to local resources on those nodes, 3) to coordinate all service instances. Service composition as resource allocation must be performed as an atomic operation using transactional techniques.

Therefore to compose an application network it is required, in first place a code distribution mechanisms that transfers application code to each node where the application servers has to be executed. In second place, mechanism to bind service programs to local resources such as memory, storage, CPU, network. In third place composing a service involves coordinating service instances. Coordination is defined by connectivity among a group of members and by rules governing interactions among those members. Rules governing interactions among service instance can be very complex, however in most applications there exist only a few simple rules of the type: "if cannot be processed here, forward to node x", "if coming from nodes x or y, forward to node z", etc. These are the rules supported by proxy caching servers as squid [Wes98], email servers [All85], IRC chat servers [Kal00]. A complete list of request routing mechanism can be found in [Bar02a]. Application network coordination is achieved by configuring how to communicate with other service instances, and which rules govern those interactions.
Software programs, which contain service instructions, have to be distributed to nodes where it has to be activated. Existing programmable infrastructures distributed code embedded in actual packets [Ten97], or using mechanisms based in file transfer protocols [Ami98][Ric98a]. There exist many mechanisms for code distribution with different levels of security and performance. It can send by unicast connections to each of different node in a TCP reliable connection with a file transfer protocol as FTP [Pos85]. Or it can be multicasted to a service multicast group using a reliable multicast transport protocol as RMP [Flo95].
These mechanisms are PUSH based, code is transmitted from an origin to destinations. The alternative is a PULL based mechanism: nodes are sent the address of a code server where they download service code in a unicast connection.

Server programs files are packages and encoded for efficient distribution. Packages are associated with a program installation utility and include information to help installation. The most widely use packages formats are Unix packages such as Linux RPM [Ewi96] and for Java applications: Java Archives JAR [SUN97], and Web Archives WAR for servlets [SUN01].
Resource Binding Mechanisms

Dynamic resource binding mechanisms are required so that services bind and unbind from resources on the fly, responding to demand variations. Application programs require permanent storage resources to save locally its code, and during program execution for data manipulation. Application programs require network capacity to provide its service and to communicate with other servers. Application programs require computational resources to process client request.

Binding service programs to local resources involves assigning resources and providing access rights: indicating which local file system directories can use, which access privileges they have, which physical memory they can use, which process identifier are assigned, which network addresses are assigned, etc. Packages containing service programs have to be decoded and installed in an appropriate local file system location with adequate access permissions. Scheduling for execution starts service programs, i.e. they are allocated a process Id and CPU slot. Resource binding mechanisms have to taken into account that several things can go wrong: more processes cannot be allocated; memory for service code can be fully utilized; initialisation parameters cannot be correct, or it cannot have adequate privileges for execution.

Binding mechanisms depend on the execution environment, some environments support run-time on the fly configuration, while others require to halt and restart services for new configurations to have effect. Application programs traditionally bind themselves to local resources invoking an operating system call at service start time. They demand memory, CPU or network capacity and are provided with some handle to its resource allocation. Without modifying service code it is required to halt and restart service programs to change its resource bindings. Another possibility is that application programs had encoded how to bind and unbind from resources at an external signal command. A solution that need not to change applications code should be virtual resource binder: application bind to them at service start
time, and virtual resource binders bind and unbind to real resource at resource agents command. If service instances were not informed about allocated local resources, they will have to scan all local resources to discover those assigned to them.

**Dynamic Service Installation**

Packages manipulation utilities provide functionality for service code installation. A review of packaging technologies for automating software deployment can be found in [Car98]. Most packages provide an indexing functionality to list locations where files have to be installed and access rights they must have; as well some kind of compression is included to decrease download time, figure 3.19. Some archive utilities, such as Linux RPM packages, have a packages dependency field that indicates whether a program requires another programs to be installed for proper functioning, and even they have compiling and linking functionality, making it ideal for distribution of source code programs. Since different services can use equal programs, providing service to different users groups. Packages are installed once, and different services differentiate by its configurations.

**Dynamic Service Activation**

Traditionally application programs bind to resources at start up. Programs are informed on which storage can use for application data and on which network interfaces they must bind to. Initialisation configuration is given to applications aas text files read by application or in command line arguments passed to applications at start-up. Different services using equal programs need to be provided a unique configuration file. As can be seen in figure 3.19, a services binds to resources by linking its code to local resources through calls to local execution environment which inform it of its name, installation location, memory requirement, local addresses, etc. The local execution environment is in charge of checking whether proposed local resources are available, and if they access to them.
**Inter-Server Communication Channels Creation**

Applications create TCP virtual circuits or UDP port pairs using sockets abstraction [Ste90]. Socket configuration parameters are: socket type, TCP or UCP, assigned port, local IP address and remote IP address. Typical server applications read such information from a configuration file at start up. Therefore to change inter-server communication channels requires stopping a server, changing its configuration file and restarting it, however application network deployment requires on-the-fly switching of communication channel. Applications code could switch communication channels on receiving an external signal as is done by squid [Wes98]. Or applications might bind to a communication abstraction that could switch its TCP virtual circuits or UDP port pairs on receiving an external signal.

**Per-Server Coordination**

Coordination is defined by rules governing relations among servers. Rules governing coordination among service instance are usually very simple application layer routing rules: “if cannot be processed here, forward to node x”, “if coming from nodes x or y, forward to
node z”. These are the rules supported by proxy caching servers as squid [Wes98], email servers [All85], IRC chat servers [Kal00]. A complete list of request routing mechanism can be found in [Bar02a]. Usually each application employs only one type of rule; therefore it is not necessary to specify it as service wide constraints.

Coordination rules can be configured at each node, or a coordination server can contain up-to-date rules that are queried by service instance. Application networks coordination rules are stored locally at each service instance. Proxy caching servers are locally configured with a set of rules of the kind “if cannot be processed here, forward to node x”, which only require local information for processing and are executed locally. Services whose nodes keep organization configuration local are provided that information individually as seen in figure 3.20.

![Figure 3.20. Per-Server Coordination Agents](image)

**Centralized Coordination Server**

Also coordination information can be centralized in a remote server. Application servers at each node query this node to find out latest coordination information or this node redirects request to most appropriate node. F.e. content distribution services required rules of the form
"if coming from nodes x or y, redirect to node z", to redirect different ranges of clients to different service nodes. Such configurations are better stored in a centralized remote server because which has global knowledge of the relative position of all clients to all servers. This is usually a DNS or an HTTP redirector server [Bar02a]. Services whose servers read coordination rules from a directory require some mechanisms to insert and update information in such directory server as seen in fig 3.21. There can be a short interval where service does not work properly, because redirection tables are not inserted/updated yet or directory servers have to be restarted, which is not desired. In this case there is no need to restart server, unless the centralized coordination server is changed too.

![Fig. 3.21. Centralized Coordinator Server](image)

**Self-Organization**

If server instances were not reported about other server instances, they might still be mechanisms to coordinate them. Servers might discover neighbour nodes that provide an equal service, and start a self-coordination process. Research projects such as LSAM [Tou98] and AWC [Zha98] are investigating how to self-organize web caching hierarchies.
3.5. Security Threats and Measures

Every element of a computer system is subject to threats. Vulnerable elements in a framework for deployment of application networks are: resources, communications and applications. Each element can be exposed to three kinds of threats: secrecy threats, elements should be kept private for correct operation of the system; integrity threats, elements should not be modified for correct operation by the system; and identity threats, elements should be authenticated and authorized for correct operation of the system.

Resource Security

Resources suffer three types of security violation:

- Secrecy: resource might not want to make public its properties and availability to unauthorised parties (or to anybody, though this case has not been consider in these work). However resource properties and availability can be determined using probes, programs which use such resources and determine its characteristics by inference.

- Integrity: Resource integrity is violated through unfair usage by programs. It is due to malicious programs or badly behaving programs (because of programmer errors or programs exceptions not caught correctly). An execution environment that controls how much resources are being consumed by each program, and kills or delay execution of them can avoid it. One execution environment which can provide such measures are “virtual machines” which execute every code line on behave of programs on the real machines after having effectuated various checks on them, among them it could be resource access check.

- Identity: Non- trusted code threat resource secrecy and integrity.
Communication Security

There exist three type of information that can suffer security threats because of being transmitted over a network: resource availability, service specifications, and deployment operations messages. Security threats are:

- **Secrecy**: access to resource availability, service specifications, deployment operations or application code can be exercise wiretapping the network.

- **Integrity**: modifications to resource availability, service specifications, deployment operations or application code can be made in messages in transit through a network.

- **Identity**: non-trusted resource agents or deployment managers can violate integrity and secrecy if not correctly identified.

Public key based authentication and encryption mechanisms are proven to solve all this security threats.

Application Security

Security threats that can affect applications are:

- **Secrecy**: Malicious host can read application code, which intended to be secret.

- **Integrity**: Malicious host can modify application code, altering its operation.

- **Identity**: non-trusted resources can threat code secrecy and integrity; therefore they should not be considered for deployment. Digital signatures can authenticate resources. Cheating resources that report some capabilities at resource discovery time can be detected if they do not provide such capabilities when a program is executed at them, i.e. a probe.

A trusted resource can still be a malicious host, since even though we authenticate it, once a program is installed and executing in it, we do not exercise control in it anymore, and is subject to secrecy and integrity threats.
<table>
<thead>
<tr>
<th>Vulnerable Element</th>
<th>Resources</th>
<th>Communication (Resource Availability, Service Specifications, Deployment Operations)</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kind of Threat</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secrecy</td>
<td>Probes determine resource availability.</td>
<td>Wiretapping.</td>
<td>Malicious resource nodes read code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>Communications Encryption.</em></td>
<td></td>
</tr>
<tr>
<td>Integrity</td>
<td>Unfair use.</td>
<td>Modification</td>
<td>Malicious resource nodes modify code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Virtual Machine with code verification.</td>
<td><em>Message Integrity Check.</em></td>
<td><em>Results integrity check.</em></td>
</tr>
<tr>
<td>Identity</td>
<td>Non-trusted code threats resource integrity.</td>
<td>Non-trusted parties threat data secrecy and integrity.</td>
<td>Non-trusted resources threat code secrecy and integrity.</td>
</tr>
</tbody>
</table>

*Table 3.6. Security Threats and Measures*
3.6. Summary and Hypotheses to Verify

In this chapter we described a model and architecture of a framework for application network deployment. First we make a hypothesis on which building blocks constitute such framework. Next we make a hypothesis on which mechanisms are more appropriate for each building block. Then we make a hypothesis on which architecture is adequate for such framework and how these building blocks map onto the architecture. Finally we study which security threats are possible and make a hypothesis on security measures that can eliminate those threats.

In the following chapter we verified such hypotheses by implementing the framework for application network deployment, and performing experiments with it.
<table>
<thead>
<tr>
<th><strong>Hypothesis to verify by implementation and experimentation</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Application networks are deployed faster that what could be done previously by manual operations.</td>
</tr>
<tr>
<td>An application network deployment framework requires these building blocks: a programmable Internet service infrastructure, resource availability, discovery and monitoring mechanisms, service specification mechanisms, resource mapping and deployment plan creation and control, and resource allocation, code distribution and service composition mechanisms.</td>
</tr>
<tr>
<td>Framework functionality is divided among deployment managers, situated at a service provider locations, and resources agents, placed at resource provider nodes.</td>
</tr>
<tr>
<td>The programmable Internet service infrastructure provides shared resources, virtualization support for multiple applications and an execution environment.</td>
</tr>
<tr>
<td>Resource availability is known by monitoring, notification or probing mechanisms.</td>
</tr>
<tr>
<td>Resource discovery and monitoring is provided by a proactive mechanisms based on multicast expanded ring searches.</td>
</tr>
<tr>
<td>Application network specifications are made up of a number of session requirements, a service demand table, and service wide constraints.</td>
</tr>
<tr>
<td>Connected placement is an appropriate mechanism for resource mapping.</td>
</tr>
<tr>
<td>A deployment control mechanisms is required so that resource allocations and service composition operations are carried out atomically on multiple remote nodes.</td>
</tr>
<tr>
<td>A local resource manager at resource node allocates resources to applications.</td>
</tr>
<tr>
<td>An HTTP based information retrieval mechanism provides code distribution.</td>
</tr>
<tr>
<td>A package installation utilities and system calls for program execution provide resource binding to permanent storage, memory and processor resources. Programs make use of other systems calls to obtain resource bindings to network and other specific resources.</td>
</tr>
<tr>
<td>Server coordination requires creating communication channels among server instances and configuring coordination rules at each node or in a centralized server.</td>
</tr>
<tr>
<td>Code authentication, and virtual machines with code verification are measures required for resource security. Party authentication, message integrity checks and communication encryption are measures required for communication security. Resource authentication, results integrity checks and encrypted program execution are measures required for applications security.</td>
</tr>
</tbody>
</table>

*Table 3.7. Hypothesis to be verified*